NPS-EC-95-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Architecture and Protocols for a Decentralized Group Membership Service for Wide-area Networks

by

Shridhar B. Shukla
David S. Neely
John Kostrivas

22 February 1995

Approved for public release; distribution is unlimited

Prepared for: National Science Foundation
Arlington, VA 22230

Naval Postgraduate School
Monterey, California 93943-5000

Rear Admiral T. A. Mercer
Superintendent

H. Shull
Provost

Shridhar Shukla was funded by the NSF RIA Grant CCR9309316
Approved for public release; distribution unlimited.

Shridhar B. Shukla

for
David S. Neely

John Kostrivas

Reviewed by:

MICHAEL A. MORGAN
Chairman
Department of Electrical and
Computer Engineering

Released by:

PAUL J. MARTO
Dean of Research

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE February 1995	3. REPORT TYPE AND DATES COVERED Final Report

4. TITLE AND SUBTITLE Architecture and Protocols for a Decentralized Group Membership Service for Wide-area Networks	5. FUNDING NUMBERS RIA Grant CCR9309316
6. AUTHOR(S) Shridhar B. Shukla, David S. Neely, John Kostrivas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-EC-95-004
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Science Foundation, CISE/CCR 4201 Wilson Blvd. Arlington, VA. 22230	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Entities cooperating as a group become simpler to construct if they possess access to a membership service to manage and administer the membership information of such groups. This report describes the architecture and design of a wide-area group membership service. Unlike any known membership service, the service is based on a completely decentralized protocol executed by a hierarchy of servers. This hierarchy permits a clear separation between the membership service infrastructure and support for application groups, permitting global scaleability. The membership protocol itself is executed by a core set of membership servers identified in a group-specific manner, permitting a separate name space, membership scope and partition handling for each group. We describe a suitable application programmer's interface and provide correctness arguments for the protocol. A working implementation of the basic membership protocol is described.

14. SUBJECT TERMS Group membership, multicast, decentralized, change protocols, network partitions.	15. NUMBER OF PAGES 31
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

TABLE OF CONTENTS

ARCHITECTURE AND PROTOCOLS FOR A DECENTRALIZED GROUP MEMBERSHIP SERVICE FOR WIDE-AREA NETWORKS

Shridhar B. Shukla
David S. Neely
John Kostrivas

Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

February 22, 1995

ABSTRACT

Entities cooperating as a group become simpler to construct if they possess access to a membership service to manage and administer the membership information of such groups. This report describes the architecture and design of a wide-area group membership service. Unlike any known membership service, the service is based on a completely decentralized protocol executed by a hierarchy of servers. This hierarchy permits a clear separation between the membership service infrastructure and support for application groups, permitting global scaleability. The membership protocol itself is executed by a core set of membership servers identified in a group-specific manner, permitting a separate name space, membership scope and partition handling for each group. We describe a suitable application programmer's interface and provide correctness arguments for the protocol. A working implementation of the basic membership protocol is described.

I. INTRODUCTION

Construction of distributed applications that require co-operation among a group of entities is facilitated by a variety of network-based services which provide clock synchronization, wide-area file systems, naming and directory information, and secure remote access. Such applications are typically also required to manage dynamic groups of entities, such as processes, devices etc., for which the membership changes due to voluntary, as well as involuntary, actions. As such applications proliferate on a typical organization's wide area network, access to a membership service (MS) to manage and administer the membership information of individual groups becomes necessary. As shown in Figure 1, each group's members can access the MS provided by membership servers (mservers) spread over an a wide area network (WAN) supporting independent, nested and overlapped groups.

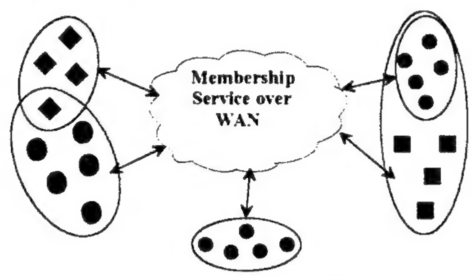

Figure 1: Membership Service and Client Application Groups

The type of membership information required depends upon the nature of the cooperation to be achieved by the members of the client groups. Examples of membership-related information are group size, members' identities, their geographical and organizational distribution, and a history of membership changes.

A membership service has typically been provided as an embedded component of toolkits for building distributed applications. Table 1 provides a summary and comparison of the known protocols for both synchronous and asynchronous environments. When the maximum delay faced by a message can be guaranteed, the environment is called synchronous. Such a guarantee cannot be provided in asynchronous environments in which a message may be delayed arbitrarily. The Internet represents an asynchronous environment. We note that many of the protocols listed in Table 1 make unique assumptions about the communication environment.

The main motivation for this work is that none of the existing approaches takes the view that a membership service is useful as an internetwork-wide service much like the name/directory services such as the Domain Name Service or X.500. In each of the cases, the protocol lacks one or more of the basic features of scaleability over WANs, portion of a range of membership services, and, in many cases, assumes network properties that are not representative of today's wide-area networks.

Therefore, the MS described here, assumes a ``best-effort" network such as the Internet, is scaleable with respect to the size/distribution/number of groups, uses network-level multicasting when available, employs a decentralized protocol with provably minimal number of phases for committing changes, and offers different qualities-of-service(QoS).

Table 1: A Summary of Existing Membership Protocols

Protocol	**Required Network Properties**	**Principle Feature**	**A**	**H**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**X**
Asynchronous Environment:												
Chang *et al.* [13]	unreliable message	token site	-	-	-	-	-	-	-	-	S	-
Bruso [14]	message diffusion	version numbers, stable storage	E	-	-	-	-	-	-	-	E	-
El Abbadi *et al.* [15]	unreliable message	virtual partitions	E	U	S	S	-	E	S	-	S	-
Verissimo *et al.* [16]	broadcast LAN	two-phase accept	-	-	-	-	-	-	-	S	E	-
Moser *et al.* [17]	ordered, reliable	ordinal numbers	-	-	-	-	-	E	-	-	-	-
Riccardi *et al.* [9]	unreliable message	reconfiguration manager	E	E	E	E	S	E	-	-	-	-
Mishra *et al.* [18]	ordered, reliable	Psync & conversations	-	-	E	E	-	E	U	-	S	E
Auerbach *et al.* [19]	multicast hardware	multicast sequences	-	E	-	-	U	S	S	-	S	-
Jahanian *et al.* [12]	unreliable message	crown prince	E	E	E	E	U	-	E	-	S	S
Golding *et al.* [20]	unreliable message	time-stamped anti-entropy	E	S	S	S	S	-	S	-	S	-
Synchronous Environment:												
Cristian [5]	bounded delay	attendance lists	-	-	-	-	S	-	-	S	E	-
Ezhilchelvan *et al.* [21]	bounded delay	time-domain multiplexing	-	-	-	-	S	-	-	S	S	-
Kim *et al.* [22]	TDMA bus	reception history	-	-	-	-	S	-	-	S	S	-
Rodrigues *et al.* [23]	exposed LAN interface	transmit-with-response	-	-	-	-	S	-	-	S	S	-

Index to Entries **S**: Supported, -: Not supported, **E**: Support possible with extensions, **U**: Unknown
Index to Columns **A**: Adaptive status monitor, **H**: Hierarchical protocol, **L**: scaLability to large groups, **M**: Multiple network support, **n**: Non blocking

This report describes a wide-area membership service to facilitate the creation and management of large, widely distributed groups for which different types of membership-related information must be maintained. Figure 2 shows the relation of the MS protocols to TCP/IP suite of internetworking protocols.

Section II describes the MS architecture and its discrete components, Section III defines and explains the protocols used by MS and Section IV describes the implementation of the core membership

protocol. In Section V we give a more formal description of the protocol and provide correctness arguments. We conclude with Section VI.

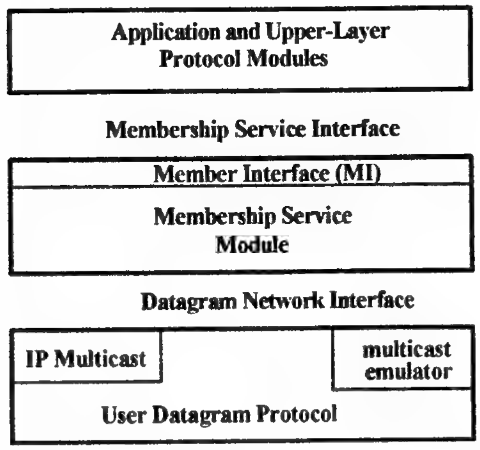

Figure 2: Position of MS in the Communication Protocol Hierarchy

II. MEMBERSHIP SERVICE ARCHITECTURE

The key to a scaleable MS is a decentralized, hierarchical architecture, designed to exploit the existing physical topology of subnetworks, networks, and internetworks upon which the distributed application process groups that the MS supports will be running. This section describes the structure and composition of the physical hierarchy of the MS and how this architecture supports application process groups.

A. COMPONENTS AND THEIR FUNCTIONS

1. Mservers And Member Interfaces

The MS is comprised of two primary entities: membership servers (mservers) organized in a physical tree hierarchy and member interfaces (MI) that represent the leaves of the tree. The mservers are primarily responsible for processing changes and providing information to the members of the physical hierarchy as well as the application process groups using the MS. Application group processes interface with the MS through an MI process running on each host computer. Each MI accepts requests for changes to or information about application groups from the individual application member processes running on the particular host computer. The MI then reliably relays these requests to the LAN mserver to access the MS. The MI receives responses from the LAN mserver and reliably propagates these responses to the application member processes that it supports. Each MI supports numerous application groups and numerous individual member processes from each application group.

Figure 3 illustrates an example logical hierarchy of mservers, MIs, and application group processes. The architecture shown is a representative configuration for a small area encompassing a single institution, such as a campus or business. The MI process remains resident on the host computer even if no applications

are running to provide quick access to the MS. The logical hierarchy shown in Figure 3 corresponds to the physical topology of networks and computers. It shows 11 departmental LANs served by as many mservers. The 11 mservers form 3 groups at the building level. At the next level (top) 3 mservers form a group to serve the entire campus. This hierarchy of mserver groups forms the MS infrastructure. Figure 4 shows the messages that are exchanged between LAN mservers, Member Interfaces and application group members.

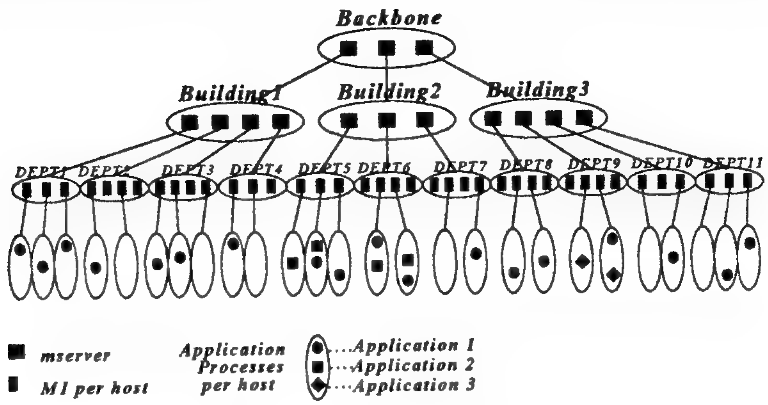

Figure 3: Logical MS Hierarchy

The mservers and MIs are administratively configured into the desired physical hierarchy by a local system administrator or cognizant authority. This configuration is expected to be semi-static, normally changing only when additions and deletions to the physical topology are made. The system administrator will assign appropriate names for each set of mservers at each level. If network-level multicasting is available, will join each set into a multicast group for efficient communication. The assignment of a set name and multicast address is accomplished when the set of mservers is created and joined together, using software calls between the MS and the kernel.

2. Failures, Partitions, And Dynamic Reformation

Mserver failures and network partitions lead to a dynamic reconfiguration of the physical structure of the MS, with the surviving mservers and MIs automatically reforming into partitioned sets. Perceived mserver failures represent virtual partitioning of the network into one or more partitioned subsets of the original set of mservers. Each partitioned subset corresponds to the subtree of the physical hierarchy in a single piece of the partition. A piece of a partition refers to all of the mservers which are still able to communicate over the non-partitioned network. Each partition of the MS reforms and continues to function, providing service to all application process groups with all members still existing in the partition. The application process groups which span the partitioned network will experience a partition in their membership. This condition will continue until the physical network partition is repaired, at which time the physical hierarchy of mservers will either administratively or automatically be reformed to the original configuration.

Once the physical hierarchy is restored, the surviving application groups will also be reformed, as per the QoS related to partition resolution chosen by the MS user at start up time. Mservers may detect failrues/partitions of the MS by continuously monitoring other mservers in the manner described later.

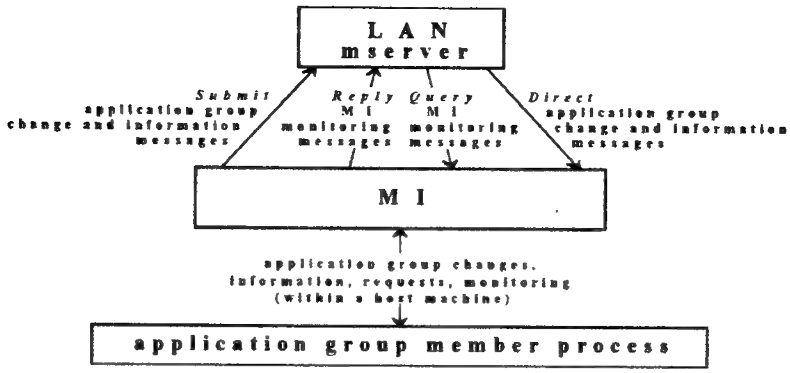

Figure 4: Messages sent by LAN Members and Member Interfaces

3. Change-Processing Core-Set

The group of mservers at each level in the hierarchy is called a change-processing core-set, with respect to a particular application group, when it is designated to be responsible for processing all membership change requests submitted by members of that application group. Every such set is also responsible for enacting changes in the physical hierarchy immediately below it. The change processing involves reaching agreement amongst all mservers in the core-set about the change submitted and propagating this change back to the application or physical hierarchy group members, who are then guaranteed to have a new view of the changed group membership.

For example, the set of mservers labeled *backbone* in Figure 3 is the core-set for the application 1 group, since this application group has members distributed on all LANs. The sets of mservers at lower levels in the hierarchy will not process membership changes for application 1, but will submit them to the core-set backbone, then relay the results back to the MIs. Similarly the mserver group labeled *building 2* serves application 2 (■). Also if the mserver for department 3 LAN fails, the mserver group labeled *building 1* will process the change to the physical hierarchy.

For each membership change request submitted to an mserver group, a coordinator is chosen. The criteria for selecting the coordinator depends on the particular type of change and how it was submitted to or detected by the mserver group. The fact that the coordinator is not a fixed member of the mserver group, but instead varies from change to change, is a powerful feature of the MS.

4. Lan Mserver Monitoring

Due to the high bandwidth, low latency, hardware multicast capability, and limited number of MIs to monitor, the mserver representing each LAN uses a simple polling scheme to conduct status monitoring

of the MIs on the LAN. Each MI on the LAN is successively polled with a *Query* message by the LAN mserver. The MI responds with a *Reply* message indicating normal status. Timeouts and retries are used to detect a non-responding MI and announce the perceived failure. Note that this monitoring emphasizes collection of status of individual MI's on the LAN. This is to be distinguished from the monitoring done by IGMP which detects if there exists a member (any) on the LAN [6].

5. Forming The Hierarchy

The final organization of mservers and MIs involves forming the hierarchy of the sets of mservers that cooperate for monitoring and change processing, with the MIs at the leaf level. As shown in Figure 3, each mserver in the hierarchy has either a set of children mservers or MIs. All mservers and MIs also have a parent mserver, except the mservers at the highest level of the hierarchy. Each mserver above the lowest level in the hierarchy has a dual membership in the "child-set" as well as the original core-set of mservers.

Having the parent mserver as a member of the child-set has two primary advantages. First, the parent mserver is part of monitoring the child set; thus, the child-set will immediately learn of the failure of the parent mserver by monitoring. Second, the parent mserver takes part in all change processing conducted by the child-set; therefore, it will learn of any changes in the membership of the child-set directly. Together, these two points ensure that "vertical monitoring" is conducted in the hierarchy. This provides the means to ensure that a failure or partition between levels in the MS hierarchy will be detected, allowing the MS to reform as necessary.

B. SUPPORT FOR APPLICATION GROUPS

The MS is responsible for managing the membership of the application groups and providing services to the application groups with features as described below.

1. Consistency

The primary service that the MS provides application groups is a consistent view of the group membership at all members, as well as a consistent ordering of changes to the membership of the group at all members. These consistency guarantees ensure that a group member either acquires the same consistent view as all other members of the group eventually, or is excluded from the membership of the group. The term "eventually" refers to the asynchronous nature of the environment, leading to delays at some sites. Using this guarantee of consistent membership at all members, the application can expect that members with the same group view number have seen the same sequence of membership changes and have the same view of the membership of the group. Using this knowledge, the application can decide to accept or reject messages from other application processes depending on the included group view number [11, 27]. The guarantee of consistent membership can be used as the foundation upon which to build many distributed applications.

The MS provides consistent ordering of membership changes to application groups by ensuring that only one change is ever processed at a time in the core-set of that application group, and that all active member processes eventually receive this change. The selected change is committed by all core-set mservers, then reliably propagated to the MIs, and finally, to the distributed application member processes. The MS provides the guarantee that an application member process either receives each revised group view or is detected as failed, and excluded from the group. In this manner, all surviving application member processes are guaranteed to have exactly the same ordering of membership changes.

2. Naming

The MS manages the names of all application groups using the MS. Application group names are guaranteed unique within a predetermined scope. When an application group is created, the software call from the application to the MS includes as a parameter a level in the MS physical hierarchy, under which the application group name will be guaranteed unique. This name-scope parameter is either the actual name of the core-set or a level number above the MI level in the physical hierarchy. For example, to guarantee an application group name of "application1" as unique under the scope of the *backbone* core-set from Figure 3, the name *backbone* or the level number 2 would be used as the name-scope parameter. The name-scope level must be at or above the core-set level for the application.

With the creation of each new application group, the name-scope parameter is checked at each level in the mserver hierarchy up to and including the name-scope level. If the name already exists, the creation of the new group is refused, and an error code is returned to the calling application. If the name is not found, then it is registered at the name-scope level of mservers and a successful group creation is reported to the calling application. When new application members at distributed locations wish to join an existing application group, a join request is submitted via the resident MI, then propagated up the hierarchy until either an mserver is located with the application name stored or the highest level in the physical hierarchy is reached and the application name is not located. If the desired application group name is located, the new member is joined into the application group through the normal change-processing sequence, and a successful join is reported back to the requesting process. If the name is not located, an unsuccessful join attempt is reported back. Through judicious use of the name-scope parameter, application names may be used freely with little concern about duplicate name usage.

3. Membership Scope Control

An additional feature provided by the MS is the ability for an application to decide at what level in the MS physical hierarchy to limit the scope of the application group. By providing a membership-scope parameter with the creation call for a new application group, the application guarantees that the span of the application's membership will not exceed that of the given core-set level in the physical hierarchy. In return, the MS is able to provide more efficient service by limiting the scope of application group name searches to

the membership-scope level and below. Instead of propagating every unsuccessful application group name search to the highest level of the MS hierarchy, the name search will cease at the membership-scope level. Without use of the membership-scope, it might be possible for a bottleneck to form at the "top" of the MS hierarchy.

4. Member Interfaces

The MI's accept application membership change and information requests from application processes and submit these changes to the mserver hierarchy for processing. When the change or information data is returned, the MI passes the data to the requesting member processes.

The MI, running on an individual host computer is capable of interfacing multiple application groups, each with multiple members, with the LAN mserver and maintains a list of all application groups it is managing as well as all member processes from these groups running on the host computer. Thus, the membership information for each application group is maintained in a decentralized, scaleable manner. When an application member process needs to communicate with another application member process on a different host, it submits a request for addressing information to the MI. The MI relays this information request to the MS, which obtains the desired information from the MI managing the desired member process, and relays the information back to the requesting MI and application member process.

5. Application Group Change Processing

As previously discussed, application group change processing begins with the submission of a change request to the host MI. This request is relayed to the core-set of the application, which conducts the mserver change-processing procedure, resulting in all core-set mservers committing the change. Each core-set mserver then reliably relays the change directive down the hierarchy to the MI, and then to the requesting application process. Timeouts and retries are again used to detect failures and partitions.

C. APPLICATION INTERFACE WITH THE MS

Table 2 summarizes the calls to the MS that provide membership service to the clients. In addition to these explicitly requested actions, the MS takes appropriate actions when failures and/or partitions occur and provides notifications to all remaining members.

III. PROTOCOL DESCRIPTIONS

A. PROTOCOL FUNCTIONS

The basic change-processing protocol uses a modified form of the three-way handshake often seen in unreliable networks for reliable message delivery. The coordinator initiates the change processing with a multicast to all group mservers, collects acknowledgment (ACK) messages from all, then multicasts a final message for all to commit the change. Timeouts and retries are used by mservers waiting to receive *ACK*s

or *Commit* messages from other mservers to ensure that continual progress is made toward completion of the change. As with the monitoring scheme, if the expected reply is not received from an mserver after the timeout period and all successive retries, then that mserver is declared failed and the failure is announced to all other mservers in the group.

Table 2: Application Interface Calls

CATEGORY	NAME OF CALL	DESCRIPTION
1	**create_group (char *group_name, char *properties)**	Creates and registers with the mservers a group with the given name with null membership. Permits the creation of a subgroup.
	set_group_attributes (char *group, char *properties, ...):	Sets the attributes of a group such as name-scope, membership scope etc.
	delete_group (char *group_name, ...):	Cleans up a group, forcing all participating members to leave.
	set_member_attributes (char *group, char *address, char *properties, ...)	Sets the attributes for the specified member of a group such as access rights.
	split_group (char *parent_group, char *child_1, char *child_2, ...)	Creates two subgroups, by spliting the parent group.
	merge_group (char *group_1, char *group_2, char *new_group, ...)	Creates a new group by merging two old groups.
2	**is_member (char *address, char *group, view *viewid, ...)**	Returns the membership status of a member for a specific group and view.
	get_view (char *group, ...)	Returns the current view scope of a group as a membership list.
	get_list (char *group, char *properties, view *viewid, ...)	Returns the members of a group with certain properties from a view.
	get_groups (int level, char *address, char *properties, ...):	Get a list of groups at a specified level.
	get_member_attributes (char *group, char *address, ...)	Gets the properties for the specified member.
	get_group_attributes (char *group, ...)	Get the properties for the specified group.
	get_sites (char *group_list, ...)	Get a list of sites on which members of specified groups exist.
	get_gsites (char *group, char *member_attributes, ...)	Get a list of group sites on which the specified group has members with specified attributes.
3	**send_msg (char *group, char *member_attributes, ...)**	Send a messge to members of a group with given attributes.
4	**join_group (char *group_name, char *my_address, ...)**	Joins a group if it exists, else creates one. Permits a new incarnation to join as an independent member.
	leave_group (char *group_name, char *address, ...)	Forces a member with given address to leave.Permits leaving groups with certain attributes.

Index for Categories: **1**:Group management, **2**: Group information, **3**: Communication, **4**: Membership

The use of timeouts and retries on change-processing messages creates a secondary but essential method of detecting mserver failures. Since mserver monitoring uses unicast messages and change-processing uses multicasts, it is possible that a network partition could occur that affected only multicast message delivery between one or more mservers. The inability of mservers to communicate all necessary data creates a virtual partition between the mservers. Without the use of this secondary detection method, it is possible that one or more mservers could be functioning perfectly well, sending the required monitoring messages, but unable to respond to change-processing messages, thus creating a deadlock situation. The timeout and retries on change-processing messages ensures that an mserver unable to communicate will be

detected failed, and the remaining mservers will be able to complete the change in a timely manner. In the event of a coordinator failure during the change processing, a distributed election is conducted and a new coordinator is elected to continue the original change. This is described later.

1. Types Of Changes

There are three primary types of membership changes processed by a group of mservers: requests, failures, and dynamic reconfigurations.

a. Requests

Requests are voluntary, planned membership changes, submitted to the group for processing by an application process or system administrator. Change requests for the MS physical hierarchy may be to *Join* to a core set, *Leave* a core set, *Split* a core set to form two new ones, *Merge* two core sets to form one, *Add_parent* to add a parent mserver to the core set and *Del_parent* to remove a parent from the core set. Physical change requests are multicast to a specific group in the hierarchy by a system configuration call, usually invoked by a system administrator during manual configuration of the MS hierarchy. Application group change requests are submitted to the resident MI process on the host computer by the application user. The MI then propagates the request to the group mserver above it in the hierarchy. The receiving group mserver queues the request to be processed when other higher priority changes have completed processing.

b. Failures

The second primary type of membership changes are detected failures. These detected failures may be the result of the actual failure of an mserver, MI, or application process, or the host machines upon which they are running. Additionally, network partitions will be perceived as failures of the partitioned mservers, and will lead to the processing of failures and reformation of the partitioned subsets of mservers and subgroups of application processes. The partitioning of the MS physical hierarchy leads to a partitioning of the application groups residing on this hierarchy. The MS automatically reforms both the physical hierarchy and the supported application groups in the event of a network partition. Failures detected or received by a group mserver are queued and processed according to their priority. Multiple failures queued at a group mserver are processed all at once, in a "batched" manner. This greatly reduces the time required to reform physical core-sets or application groups.

c. Dynamic Reconfigurations

The final type of changes are the result of automatic actions taken by core-sets of mservers. This type of dynamic reconfiguration occurs when new members join an application group, causing the span of the application group to increase beyond that presently covered by the current application core-set. In this event, the application core-set must be moved from the present level in the physical hierarchy to a higher level covering the new span of the application. This new level must be at or below the name-scope

and membership-scope levels of the application group, if these levels were designated when the application group was created. The MS automatically moves the application core-set to the new level. In a similar manner, the departure of application member processes may lead to a reduced span of the application. An application core-set must have at least two mservers with application members in their subtrees; otherwise, there is no need to have the application core-set at this level in the hierarchy. If the application core-set is reduced to only one mserver supporting an application, the application core-set will automatically move down to the child-set of this mserver.

The repositioning of an application core-set is initiated by the set of mservers detecting the need to move the application core-set. Messages are exchanged between the old and new core-sets and a change involving the join or departure of the instigating application member is processed along with the change in application core-set level by both core-sets. After committing the changes, the internal state of all mservers in both core-sets is changed to reflect the new application core-set level.

2. Ordering And Priority Of Change Processing

A key issue associated with processing membership changes is the ordering of changes committed by the core-set. As previously described, to guarantee consistent ordering of membership changes at all mservers in the group, only one change may be committed at a time. However, it is possible that more than one membership change may be submitted to or detected by the group at one time. Each receiving or detecting mserver in the group will attempt to become the group coordinator and initiate the change it received or detected. These multiple change initiation attempts are referred to as "virtually simultaneous", since they have all been initiated before the group has reached a consistent and uniform decision on the current change to process.

To resolve these virtually simultaneous changes and select only one change to be processed, a priority scheme is used. This scheme uses the type of change and the unique group id (rank) of the subject of the change to decide which change will be processed by the group. The subject of a change refers to the member whose membership status has changed. The highest priority is given to any current change being processed by the group; that is, a change which is in progress at an mserver (i.e. an *ack* to the *initiate* has been sent). It is essential that such a change progresses to completion at all group mservers; otherwise, the possibility of inconsistent membership views exists if some mservers commit the change while others do not. The next lower priority is that physical hierarchy changes always have priority over application group changes. This is because it is important to ensure a complete and whole MS infrastructure before attempting to change the membership of an application group using the MS. Once these decisions have been made, the priority of the change is determined by the rank or age of the subject of the change in the group. The only exceptions to this rule are for the failure of the coordinator of the current change or a *Join*. The failure of the coordinator of a change in progress, has priority over otherwise equal status changes. A newly joining

mserver or member will not have an associated rank until after the join is completed. For this reason, the network address of the joining member is used instead of a rank number to decide priority among *Joins.* The final rule used to determine the priority of virtually simultaneous changes is applicable when changes are submitted to the core-set by different application groups with identical subject rankings in each group. In this case, a tie-breaker is needed, and the ranks of the coordinators in the group are used to decide which change will be processed.

A. CHANGE PROTOCOL

The basic change-processing protocol consists of two phases: the Initiate and Commit phases. A timeline for this protocol is shown in Figure 5. In the Initiate phase, the coordinator multicasts an *Initiate* message to all mservers in the group. The group mservers respond with *ACKs*, acknowledging reception of the *Initiate* message. When the coordinator has received all the acknowledgments, the second phase of the protocol begins with the coordinator sending the *Commit* message. This message indicates to members of the group that is safe to commit the change. Phase I is achieved through a reliable schema. The coordinator sends the *Initiate* message a predetermined number of times if one or more group mservers do not reply. After that, it assumes that the mserver(s) that did not reply have failed.

B. CHANGE PROTOCOL WHEN COORDINATOR FAILS

The two phase protocol is not sufficient in case coordinator of a change fails while processing a change. As shown by Riccardi and Birman [9], a three phase protocol is required. After the coordinator of the current change fails, and its failure is detected by a member of the group, a three phase protocol is initiated, with the election of the new coordinator as the first phase. Figure 6(a) illustrates the three phase election and change processing protocol. In the election phase only those members that have finished phase one of the original change protocol participate.

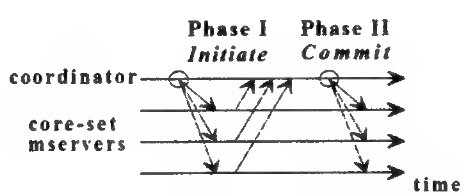

Figure 5: Basic Two Phase Change Protocol

When the new coordinator is elected, it knows the status of each member with respect to the change, due to a status broadcast during the election phase. If at least one mserver has finished the change (committed) it means that the old (and detected failed) coordinator had already collected all *ACKs* and had started the *Commit* phase. So the new coordinator, knowing that phase one was completed, can continue

with the final phase of the change, instead of restarting phase one. This compressed three phase protocol is shown in Figure 6(b).

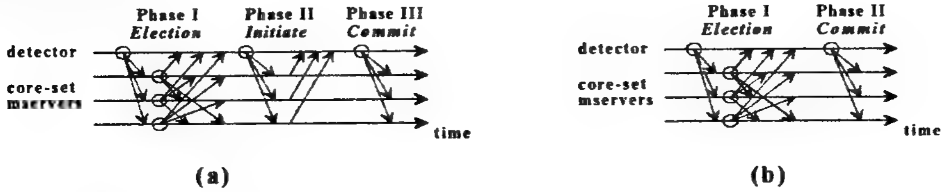

Figure 6: Three Phase (Election and Change) and Compressed Three Phase Protocol

A simplified algorithm for the two phase and three phase algorithm is shown in Figure 7. Only the important arguments are shown. All sends and receives of messages are done with timeouts. This way the protocol does not block and takes appropriate actions in case of no response. In line 5, the term "reliably" implies that a message is sent and specific answers are expected from all members in a specific time interval. If some or all of them do not reply, a number of retries are attempted. After the retries and the last timeout expire, the sender assumes the non responding mservers to have failed. In line 11, the same function is called recursively to process the failure(s) of the member(s) that did not reply. In line 8, the second phase is executed by sending a commit message. Then the coordinator has to make the change to its internal state. Lines 13 through 25 are devoted to the non coordinator. Since this code is executed everywhere, it contains both cases (coordinator, non-coordinator), and the if statement in line 3 decides which part of the code should execute.

Lines 17 through 25 form the three phase protocol in case the coordinator of the current change fails. Lines 17 through 21 refer to the member(s) that detected the failure of the current coordinator. Lines 22 through 25 refer to the member(s) still not detect the coordinator's failure but wait for a *Commit* message. Both sides triggered by the *coord_fail* message, start collecting status of the rest of the members. Then in lines 20 and 24 an election of a new coordinator is done. The lowest rank among the survived and responded mservers gets elected and all members go to process the new (and of higher priority) change. If the current coordinator does not fail, all members commit the change, updating their internal state, in line 27.

C. PARTITION RESOLUTION PROTOCOL

As part of the processing of multiple failures caused by a network partition, a group is often partitioned into two or more subsets. After the membership changes in each subset stabilize, these sub-core-sets attempt to reform into the original group by sending messages to the other subsets of mservers. Since the sub-core-sets still share the same multicast address, once the network partition is mended, the other sub-core-sets receive these reformation messages. Upon learning of the existence of a sub-group from the original group, the partitioned subsets of mservers reform into the original group automatically. In addition to reforming the physical group, all application groups which were partitioned and are still functioning are also

reformed. The reformation process for both physical core-sets and application groups merges the currently existing membership of each, taking the union of all subsets or subgroups, and making the reformed group or application group membership the current view. In the event that the network partition is not repaired in a predetermined period of time, the partitioned subsets of mservers will abandon their attempts to reform the original group, and will create a new multicast group with only the current group mserver included.

```
/*membeship change protocol */
process_change (type of change, subject)
    if coordinator
            /* start phase I */
            send agree to group reliably
            receive acks
            put members that did not reply, on fail list
            send commit message
            commit the change
            if fail list is not empty
                        process_change (fail, first in fail list)

    else    /* non -coordinator */
            receive agree msg
            send ack to coordinator
            wait for commit
            if commit is not received
                        send coord_fail msg broadcasting status
                        collect status from other members
                        determine new coordinator
                        process_change (fail, old coordinator)
            else if coord fail msg is received
                        broadcast own status and collect status from other members
                        determine new coordinator
                        process_change (fail, old coordinator)
            else        /* commit received */
                        commit change
```

Figure 7: The Membership change protocol (two phase - three phase)

If the group partitions, the application groups that span the partition, also experience a virtual partition. These partitions are handled using the following two rules.

- Keep alive any partitioned subgroups that meet a certain condition specified by the user. Any subgroups which do not meet the condition are terminated.
- Partitioned subgroups attempt to find and merge with other partitioned subgroups that have a certain user-specified property.

By combining these two rules, every possible combination of partition handling methods can be produced. The first rule determines who survives, and the second rule determines who will attempt to merge. Each rule can also combine multiple parameters to provide very specific and flexible methods of handling partitions. For example, all subgroups larger than a size of three which contain a particular member type could be permitted to survive and merge with subgroups larger than half of the original group size and containing another particular member type. Note that all partitions of the group necessarily survive network partitions.

In the event that the partitions of mservers are unable to restore communications, the reformed subsets are converted to completely independent core-sets. Since all core-sets of mservers must have a unique name and multicast address, some method must be used to automatically obtain these unique values. To obtain a unique name, each sub-group appends a unique suffix to the original group name. This suffix value must be automatically derived by each partitioned subset of mservers independently, and with a guaranteed unique value for all partitioned subsets. The most readily available attribute that all subsets can use to obtain a guaranteed unique name is the original group identity (gid) of a significant mserver remaining in each partition. The lowest mserver gid of the mservers remaining in each partition is appended to the original group name. In this manner, all partitioned core-sets are guaranteed a unique group name. However, all partitioned core-sets are still easily identifiable as subsets of the original group, which simplifies the task of manually reconfiguring the physical hierarchy when the network is repaired. Once a unique name is obtained, traffic on the same multicast address can be easily filtered by the individual sub-core-sets.

IV. IMPLEMENTATION

The use of multicast message delivery is essential to the efficient and scaleable operation of an MS. In network environments where IP multicast is not available [7], the MS must use point-to-point messages. So a *multicast emulator* was implemented to provide with the same interface to mservers running either in an IP-multicast-capable or a unicast environment. Figure 8 shows how the mcaster is embedded in the communications among mservers with IP multicast capability and unicast only capability. Each mserver has two sockets. One of them is unicast dedicated to monitoring and the other can be either a multicast or a unicast if IP multicast is not available on the LAN. If this socket is multicast mserver uses pure IP multicasting to send and receive group messages. If it is unicast, the mserver communicates with the group through mcaster. Figure 8 shows the two cases where the sender is on a multicast and on a unicast capable host.

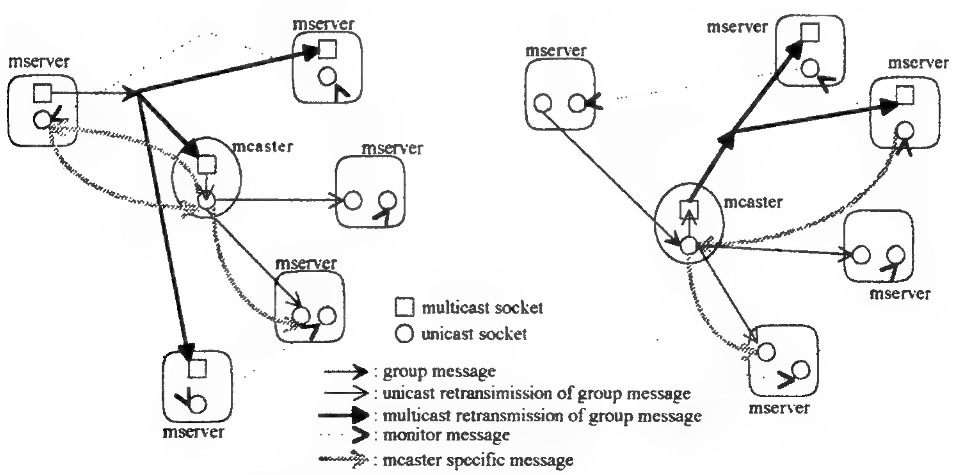

Figure 8: Communication Among Mservers

The 2-phase and three-phase membership change protocol described earlier, has been implemented and tested over a campus WAN of IP-multicast and non-multicast LANs.

V. CORRECTNESS ARGUMENTS

In this section we present the correctness arguments to show that the structure of the protocol and the actions it takes at each mserver lead to the desired attributes of the service itself.

The focus of the argument is on a core set of mservers as a group and refer to individual mservers as members of the group. It is shown that changes to the membership of this group are made to in order to maintain consistency of the membership information. Application group membership consistency can be argued about in an identical manner. The assumptions made and the terms used and their specific implications in the correctness arguments are described first. This is followed by the correctness criteria and a summary of the actions that the protocol takes to maintain the membership knowledge accordingly. Finally, key statements about different aspects of the protocol are proved.

A. ASSUMPTIONS

As described previously, an asynchronous communication environment is assumed to exist, providing an unreliable message delivery capability with an unbounded delay, as in the present *best-effort* Internet. Thus, network failures that include dropped messages and network partitions are permitted. All member failures are assumed to be crash or fail-stop [5, 9, 10, 11]. In such conditions, failures can only be perceived, and both actual member failures and network partitions lead to perceived failures of the members. For this reason, every perceived failure is processed as an event that partitions the group. Partitions of the membership of a group are assumed to be acceptable to the user of the membership service, who may make QoS selections to determine how partitioned groups will continue to function, as described in earlier chapters. Unlike many other membership protocols, majority-based decisions are not used by this MS to ensure that only a single partition survives; instead, complete agreement is required among all surviving members that can communicate, leading to the possibility of separate, functioning partitions of any size. Continuous changes to the membership are allowed; however, the changes are committed one at a time, and with a specific order in each partition.

B. TERMS AND DEFINITIONS

The specific terms and implications of their use in the correctness arguments described later are listed below.

1. Change Events

The events that cause a change in the membership are: explicit join and leave requests by members, perception of failure by the monitoring of members by other members, and suspicion of failures resulting from member or network failures which lead to a lack of response during change processing.

2. Change Event Priority

Every change event has an associated priority to enable ordering of virtually simultaneous changes. Failures have a higher priority than voluntary joins or departures. Priority of a failure or departure event is the rank, or seniority, of the failed member in the group. The most senior member always has a rank of 0. When two or more members initiate a change simultaneously, the coordinator initiating the higher priority change, as determined by the rank of the subject of the change, prevails. In virtually simultaneous joins, the subjects do not yet have a group rank, so the network address of each subject is used in place of the rank. The subject with the lower network address will be interpreted as having a higher temporary rank, and therefore will have a higher priority, joining the group first.

3. Isolation

A member that perceives another member as faulty ceases all communication with that faulty member. This leads to the member perceived as faulty also determining that the other member is faulty, since no communications are received.

4. Gossip

A member that isolates another member gossips about the isolation in the subsequent communication it has with every other group member. Thus, in the absence of any other failures, a multicast following an isolation leads to the whole group isolating the member that was perceived faulty by the sender of the multicast.

5. Group View

This term denotes the ordered membership list maintained by each member m_i, and is denoted as $View_x(m_i)$, where x denotes the view number.

a. Definition

The group view at a member is the set of members that are perceived to be part of the group. It is ordered with respect to the seniority of members in the group and has an integer, called a view number, associated with it.

b. Remarks

Every membership change alters the number of members in the view at a member and leads to the installation of a new view identified by the next higher view number. The number of members in the group may change by more than one in a single view change. The rank of a member denotes its seniority in the group, with the most senior member having rank 0. Identical views imply identical membership as well as ranks.

6. Group Partition

Let G denote the set of all possible potential and current members of a group. A *partition* P of G is defined below.

a. Definition

P is a subset of the all members' set G, such that $\forall m_i, m_j \in P$, if $\text{View}_x(m_i)$ and $\text{View}_x(m_j)$ are defined, then $\forall m_k \in \text{View}_x(m_i)$: $m_k \in \text{View}_x(m_i) \Leftrightarrow m_k \in \text{View}_x(m_j)$, and all members have the same rank in the two views.

b. Remarks

The current view associated with partition P is denoted View_P, and the partition containing m_i is denoted $P(m_i)$. Thus, all members in a partition must have identical views. However, it is possible that there exists an m_k outside a partition, but still in every member's view for that partition. Such partitions are called *unstable partitions*. The MS protocol treats such a partition as legal, and eventually removes m_k from the views of all members of the partition. When no such m_k exists for a partition, the partition is called *stable*. Both, perceived network and member failures, lead to the creation of group partitions in asynchronous environments.

7. Group Membership Protocol

Using the definitions of the terms above, a protocol that solves the Group Membership Problem (GMP) is defined as below.

a. Definition

A protocol solves the GMP correctly if <u>every</u> change event results in <u>group partitions</u> <u>eventually</u>.

b. Remarks

The above definition of a correct solution of the GMP requires it to satisfy distinct properties corresponding to the underlined conditions in the definition above.

- **E1** This property, arising from the condition of <u>every</u>, requires that a change event observed by a member is processed despite other virtually simultaneous change events and failures during protocol execution, including that of the coordinator. The only situation in which a change event is not processed is in case of catastrophic occurrences in which all the members with knowledge of the change event suffer real failures.
- **E2** This property, arising from the condition of <u>eventually</u>, permits the processing of a change event to be suspended temporarily; however, it requires that the resulting view is always installed at all members of the partition before the change event occurred and after only a finite number of changes are allowed to take place.
- **GP** This property, arising from the condition of <u>group partitions</u>, implies identical views at all members of each partition. As per the protocol described, all partitions resulting from change processing always become stable.

Requirements imposed by the **E1** and **E2** properties satisfy the condition commonly known as *liveness* in distributed systems and those imposed by the **GP** property satisfy *safety* [5, 24, 25]. Thus, the uniqueness of views and identical ordering of changes at all operational members is guaranteed by **GP**.

C. REMARKS ON THE PROTOCOL STRUCTURE

Unless specified otherwise, the term failure is assumed to imply a perceived member failure that may have been caused by either a network failure or a member's failure. Any of the members may initiate a change when it perceives a change according to the change events described earlier. The change initiator is called the coordinator for that change and carries out the membership change protocol of Figure 7. The non-coordinator's actions consist of sending the *ACK* message and committing the change. Due to the possibility of other changes occurring during a change processing, both the coordinator and non-coordinators must take additional actions. Depending upon the message received by the coordinator as it collects the *ACKs*, it switches to a higher priority change, enters an election, or delays the change it is coordinating due to a previous change that may not yet have completed.

D. CORRECTNESS ARGUMENTS

Based on the protocol detailed description given earlier, a proof is presented that shows that the MS protocol has all the properties as identified above for a correct solution to the GMP. Also shown is that a more refined solution to the GMP defined earlier by Ricciardi and Birman [9] is possible.

1. Claim 1

Change event processing always completes at both the coordinator and the non-coordinator except when all members, including the coordinator, with knowledge of the change fail.

2. Proof

Consider a change event *change(subject, coordinator)* initiated in *P*.

a. At The Coordinator

Although the coordinator makes multiple attempts to deliver the *Initiate* message to all perceived members of *P*, it does not require a predetermined number of them to respond before it sends a *Commit* message. If the coordinator switches to a higher priority change before it sends a *Commit* message, the information about the old change is saved. The old change is reinitiated by the coordinator after all higher priority changes complete.

b. At The Non-Coordinator

The non-coordinators waiting for the *Commit* of the lower priority change, switch to the higher priority change, when the corresponding *Initiate* is received.

If the coordinator fails, at least one member times out on the *Commit* message and starts an election. The highest rank member with the change active is elected to resume the change. The fact that

the election is conducted among those with knowledge of the change ensures that the change completes even if the coordinator and the only members to have committed the change fail. This takes care of the invisible commits described by Ricciardi and Birman [9].

3. Claim 2

In any partition, either only one change event proceeds to the commit phase, or members reaching the commit phase for different change events form separate partitions.

4. Proof

Initially, all members have identical views of the membership (definition of a partition). In the set of all potential change events, there exists a unique priority order due to the uniqueness of ranks, which order failures and departures, and network-level addresses, which order joins. This permits every member receiving multiple *Initiate* messages before receiving any *Commit* message to switch to the highest priority change that will install the next view. Overlapping of *Initiate* messages to install successive views with different view numbers is not possible, because every member accepts only the highest priority *Initiate* to establish the current change.

Suppose a member receives a *Commit* message for the current change that will change the view number from x to $x+1$. Suppose this mserver then receives a higher priority change that also corresponds to a view number change from x to $x+1$. It is guaranteed that the coordinator sending the higher priority *Initiate* appears in the gossip accompanying the received *Commit* message. This happens because the coordinator of the lower priority change will have timed out on the coordinator for the higher priority change and isolated it before generating a *Commit* message. This ensures further partitioning.

5. Claim 3

If the coordinator fails after sending the commit message, a two-phase protocol consisting of an election followed by a commit can solve the group membership problem correctly.

6. Proof

Begin by proving the contrapositive statement:

A two-phase protocol consisting of an election followed by a commit cannot solve the GMP correctly if the coordinator fails before sending the commit.

If the coordinator fails before sending the *Commit* message, it is possible that one of the members has not yet received the *Initiate* message for the change. This member would respond in the election with a *Coord-Fail* message that announces that it is not aware of the change for which the election has been started. This member must receive an *Initiate* message before it can commit the change for which the coordinator failed. If the *Coord-Fail* message is used to start the change in place of a separate *Initiate*

message, and only a *Commit* message is sent to complete the change, then the **GP** property can be violated, as shown in the example below.

Consider a partition consisting of members m_i, m_j, m_k, C_a, and C_b. Let C_a initiate change "*a*" by multicasting $Initiate_a$, which is received only by m_i due to network failures. C_a fails immediately after sending $Initiate_a$, and this failure is perceived by m_i, which then starts an election by multicasting $Coord\_Fail_a$. m_j and m_k participate in the election, but C_b does not because it has failed. However, before failing, C_b starts another higher priority change by multicasting $Initiate_b$, which reaches only m_j due to network failures. Since change "*b*" is a higher priority change, m_j drops change "*a*" as the current change. At this point, m_j perceives C_b failed and starts an election by multicasting $Coord\_Fail_b$.

Throughout this time, m_i waits to hear C_b's response to the election for change "*a*", which will not arrive due to C_b's failure before $Coord\_Fail_a$ reaches it. Eventually, m_i times out in the election, determines that it must be the winner, and assumes the responsibility for completing change "*a*". Using the compressed 3-phase protocol of Figure 6(b), m_i skips the $Initiate_a$ and commits change "*a*" and multicasts $Commit_a$ to the group with gossip about C_b's isolation. If the $Commit_a$ reaches m_j and m_k after they have switched to change "*b*" due to the $Coord\_Fail_b$ message, they will quietly discard the $Commit_a$ message due to its lower priority. Thus, m_i will have committed change "*a*", whereas m_j and m_k will never commit it. This inconsistency violates the **GP** property and makes the use of a two-phase protocol incorrect in this situation. Thus, the contrapositive statement is proved. This will not happen if m_i used all the three phases.

The contrapositive statement proves Claim 3 above. It should be noted that the failure of the coordinator after sending the *Commit* message with simultaneous failures of all members that receive the *Commit* message is equivalent to the coordinator failing before sending the *Commit* message. It is not possible to differentiate between these two situations, thus the change must be completed in three phases. In the protocol described in this report, the three phases are the broadcast election, initiate, and commit phases. Thus, the elected coordinator is required to complete the change with a *Commit* message if some member that had committed the change participates in the election, permitting a two-phase processing of the coordinator failure. Otherwise, the elected coordinator simply reinitiates the change, providing a three-phase processing of the original change.

7. Theorem

The proposed group membership protocol is safe and live.

8. Proof

The *liveness* properties E1 and E2 follow directly from Claim 1. The *safety* **GP** property follows from Claim 2 and 3.

VI. CONCLUSIONS AND FUTUREWORK

A. CONCLUSIONS

This report presented a globally scaleable, fully decentralized group membership service which provides the framework for distributed applications of any size and distribution. A complete description of the architectural design and protocol specifications was presented, and a complete implementation of the basic membership is described.

The most significant contribution of the group membership service described herein is the definition of the hierarchy and associated functions. The MS provides a consistent suite of services to client applications distributed on any scale, from a single LAN to the worldwide internetwork. No other membership protocol or service provides this capability. Other noteworthy contributions include the decentralized and efficient nature of the MS, the concept of providing numerous user-selectable Quality of Service options to tailor the MS to the needs of each client application and a programmer's interface permitting flexible access to the membership service.

B. FUTURE WORK

Although a great deal of work was accomplished in the design and protocol implementation for the MS described in this report, much more work remains to be done. First, to demonstrate that the MS is truly scaleable to global proportions, a working implementation of the complete MS must be developed and installed on progressively larger scales. Second, complete performance analysis of the operation, overhead, network constraints, service latency, and functionality of the MS must be accomplished. Third, a complete formal specification of the protocols used by the MS must be accomplished, with a reachability analysis conducted to demonstrate formally correct operation. Finally, the MS must be extended to take advantage of the reliable, high-speed networks with newer network-level multicasting options, which are currently being deployed.

LIST OF REFERENCES

1. K. P. Birman, "The process group approach to reliable distributed computing," Communications of the ACM, no. 12, vol. 36, December 1993.

2. F. Cristian, R. Dancey, and J. Dehn, "Fault-tolerance in the advanced automation system," The 20th International Symposium on Fault-tolerant Computing, pp. 6-17, June 1990.

3. L. L. Peterson, N. Buchholz, and R. D. Schlichting, "Preserving and using context information in interprocess communication," ACM Transactions on Computer Systems, vol. 7, no. 3, pp. 217-246, August 1989.

4. D. Powell, M. Chereque, D. Drackley, "Fault-tolerance in Delta-4," Operating Systems Review, vol. 25, no. 2, pp. 122-125, April 1991.

5. F. Cristian, "Agreeing on who is present and who is absent in a synchronous distributed system," Proceedings of the 18th International Conference on Fault Tolerant Computing, Tokyo, Japan, pp. 206-211, 1988.

6. S. Deering, "Host extensions for IP Multicasting,"Technical Report De89, Internet, Network Working Group, RFC 1112, August 1989.

7. S. Deering,"Multicast routing in a Datagram Internetwork", PhD thesis, Stanford University, December 1991.

8. S. Zabele and R. Braudes, "Requirements for multicast protocols," Internet, Network Working Group, RFC 1458, May 1993.

9. A. M. Ricciardi and K. P. Birman, "Using process groups to implement failure detection in asynchronous environments," ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, pp. 341-353, August 1991. Also available as TR91-1188, Dept. of Computer Science, Cornell University.

10. R. D. Schlichting and F. Schneider, "Fail-stop processors: an approach to designing fault-tolerant computing systems," ACM Transactions on Computer Systems, vol. 1, no. 3, pp. 222-238, August 1983.

11. K. P. Birman and T. A. Joseph, "Reliable communications in the presence of failures," ACM Transactions on Computer Systems, vol. 5, no. 1, pp. 47-76, February 1987.

12. F. Jahanian and W. Moran Jr., "Strong, weak and hybrid group membership," Proceedings of the Second Workshop on the Management of Replicated Data, Monterey, California, pp. 34-38, November 1992. Also available as Technical Report RC 18040 (79173) 5/28/92, IBM Research Division, T. J. Watson Research Center, 1992.

13. J. M. Chang and N. F. Maxemchuk, "Reliable broadcast protocol," ACM Transactions on Computer Systems, vol. 2, no. 3, pp. 251-273, August 1984.

14. S. A. Bruso, "A failure detection and notification protocol for distributed computing systems," Proceedings of the 5th International Conference on Distributed Computing Systems, pp. 116-123, May 1985.

15. A. El Abbadi, D. Skeen, and F. Cristian, "An efficient fault-tolerant protocol for replicated data management," Proceedings of the 4th ACM Symposium on Principles of Database Systems, pp. 215-229, 1985.

16. P. Verissimo and J. A. Marques, "Reliable broadcast for fault-tolerance on local computer networks," Symposium on Reliable Distributed Systems, pp. 54-63, October 1990.

17. L. E. Moser, P. M. Melliar-Smith, and V. Agrawala, "Membership algorithm for asynchronous distributed systems," Proceedings of the 11th International Conference on Distributed Computing Systems, pp. 480-488, 1991.

18. S. Mishra, L. L. Peterson, and R. D. Schlichting, "Consul: A communication substrate for fault-tolerant distributed programs," Technical Report TR 91-32, Department of Computer Science, University of Arizona, 1991

19. J. Auerbach, M. Gopal, M. Kaplan, and S. Kutten, "Multicast group membership management in high speed wide area networks," Proceedings of the 11th International Conference on Distributed Computing Systems, pp. 231-238, 1991.

20. R. A. Golding and D. D. E. Long, "The performance of weak-consistency replication protocols," Technical Report ucsc-crl-92-30, Department of Computer Science, University of California at Santa Cruz, July 1992.

21. P. D. Ezhilselvan and R. de Lemos, "A robust group membership algorithm for distributed real-time systems," Proceedings of the Real-Time Systems Symposium, pp. 173-179, 1990.

22. K. H. Kim, H. Kopetz, K. Mori, E. H. Shokri, and G. Gruensteidl, "An efficient decentralized approach to processor-group membership maintenance in real-time LAN systems: The PRHB/ED scheme," Symposium on Reliable Distributed Systems, pp. 74-83, 1992.

23. L. Rodrigues, P. Verissimo, and J. Rufino, "A low-level processor group membership protocol for LANs," Technical Report Oct. 1992, Technical University of Lisbon, Portugal, INESC, 1992.

24. J. Misra and K. M. Chandy, Parallel Program Design - A Foundation, Addison- Wesley, New York, New York, 1989.

25. G. Andrews, Concurrent Programming - Principles and Practice, Benjamin/ Cummings, Redwood City, California, 1991.

26. D. Comer and D. Stevens, Internetworking with TCP/IP, Vol. I: Principles, Protocols, and Architecture, 2nd edition, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

27. K. Birman, A. Schiper and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", ACM Transactions on Computer Systems, pp. 272-314, 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Dudley Knox Library, Code 052 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	National Science Foundation CISE/CCR 4201, Wilson Blvd. Arlington, VA. 22230	2
5.	Professor Shridhar B. Shukla, Code EC/Sh Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	10
6.	LT David S. Neely P.O. Box 63 Arnold, California 95223-0063	1
7.	LTJG John Kostrivas 310, Hatten Rd. Seaside, California 93955	1